Pascal Pons

Follow

SOLVING CONNECT FOUR

Part 11 – Optimized transposition table

Our initial implementation in part 7 of transposition table is using 64bits entries to store 56bits keys and 8bits values. The code used in this tutorial was using a 64MB table of about 2^23 entries.

We will implement in this section an optimized verion of it that is more memory effective with configurable key and value size.

Storing partial key and Chineese remainer theorem

Let's first look at the implementation:

```
class TranspositionTable {
  private:

  static const size_t size = 1 << 23 + 9; // size of the transpositin table must be odd, preferably prime number
  typedef key_t uint32_t;
  typedef value_t uint8_t;

  key_t *K;     // Array to store trucated version of keys;
  value_t *V;   // Array to store values;

  size_t index(uint64_t key) const {
    return key%size;
  }

  public:
  // removed constructor, destructor for lisibility.

  /**
   * Store a value for a given key
   * @param key: must be less than key_size bits.
   * @param value: must be less than value_size bits. null (0) value is used to encode missing data
   */
  void put(uint64_t key, value_t value) {
    size_t pos = index(key);
```

```
    K[pos] = key; // key is possibly trucated as key_t is possibly less than key_size bits.
    V[pos] = value;
  }

  /**
   * Get the value of a key
   * @param key: must be less than key_size bits.
   * @return value_size bits value associated with the key if present, 0 otherwise.
   */
  value_t get(uint64_t key) const {
    size_t pos = index(key);
    if(K[pos] == (key_t)key) return V[pos]; // need to cast to key_t because key may be truncated due to size of key_t
    else return 0;
  }
};
```

At a first glance we are almost doing the same as before, with the only exception that the type key_t is now 32 bits, smaller than the 49 bits needed to store a full position key.

Indeed we use the same simple transition table with modular indexing, but we will only store the last k bits of the key, k being smaller than the key size (k is 32 in the above code). The last k bits are in fact the key modulo 2^k. We store it in a table of S entries (S being odd) at position key modulo S. S and 2^k being prime each other, by the Chineese remainer theorem, the pair (key%S, key%2^k) can only map to a single integer “key” lower than S*2^k.

So if we select carefully the number of bits k and the size of the table we can guarantee that there is no collision of two different keys ending up in the same slot of the table and having the same trucated value as long as S*2^k is larger than the maximal key. We can also express this constraint as the number of bits of the key need to be less or equal than the number of bit stored bits of the key plus log2(S).

The acual implementation is a little more complex as it also allows to have flexible configuration of the key, value and table size as class template.

Case of the 7x6 connect 4 board

A 7x6 connect position key is 49 bits. We can use efficient 32bit storage of the key and 8bit storage of values as long as the transposition table size is an odd number greater than 2^17. In practice for this tutorial the size of the transposition table has been selected as the smallest prime number above 2^23, leading to a transposition table size of about (4+1)*2^23 bytes = 40MB.

Benchmark

The new improved transposition table allowed to reduce the memory need from 64MB to 40MB, while keeping the same number of entries. The new implementation, leveraging only primitive integral types is a little faster than the previous one. The reduced memory footprint also speeds up marginally the processing time. Overall the solver is a little less than 10% faster on our tests:

Solver	Test Set name	mean time	mean nb pos	K pos/s
Optimized transposition table (strong solver)	End-Easy	4.722 µs	54.93	11,630
Optimized transposition table (strong solver)	Middle-Easy	39.90 µs	517.4	12,960
Optimized transposition table (strong solver)	Middle-Medium	3.736 ms	48,450	12,970
Optimized transposition table (strong solver)	Begin-Easy	275.5 µs	3,693	13,400
Optimized transposition table (strong solver)	Begin-Medium	113.4 ms	1,459,000	12,870
Optimized transposition table (strong solver)	Begin-Hard	5.667 s	72,490,000	12,790
Optimized transposition table (weak solver)	End-Easy	3.407 µs	31.46	9,235
Optimized transposition table (weak solver)	Middle-Easy	52.21 µs	618.8	11,850
Optimized transposition table (weak solver)	Middle-Medium	1.926 ms	23,840	12,380
Optimized transposition table (weak solver)	Begin-Easy	2.013 ms	25,530	12,680
Optimized transposition table (weak solver)	Begin-Medium	51.71 ms	671,200	12,980
Optimized transposition table (weak solver)	Begin-Hard	3.380 s	42,090,000	1,2450

Tutorial plan

SOLVING CONNECT FOUR

Updated: October 22, 2017

Previous | Next